SQL Programming Views, Stored Procedures, Functions

Overview

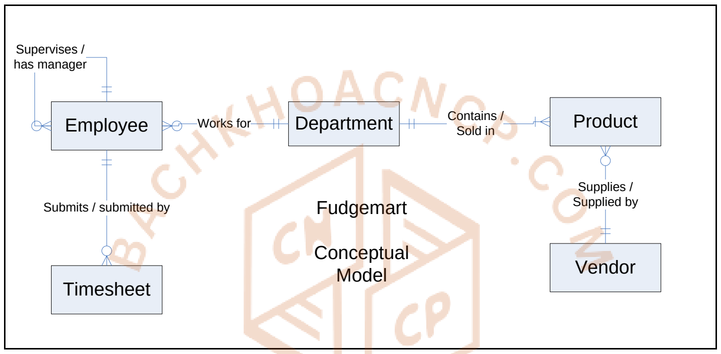
In this lab, we will use the following concepts from class:

- Views.
- Stored Procedures
- Functions
- The Execute command to run a stored procedure

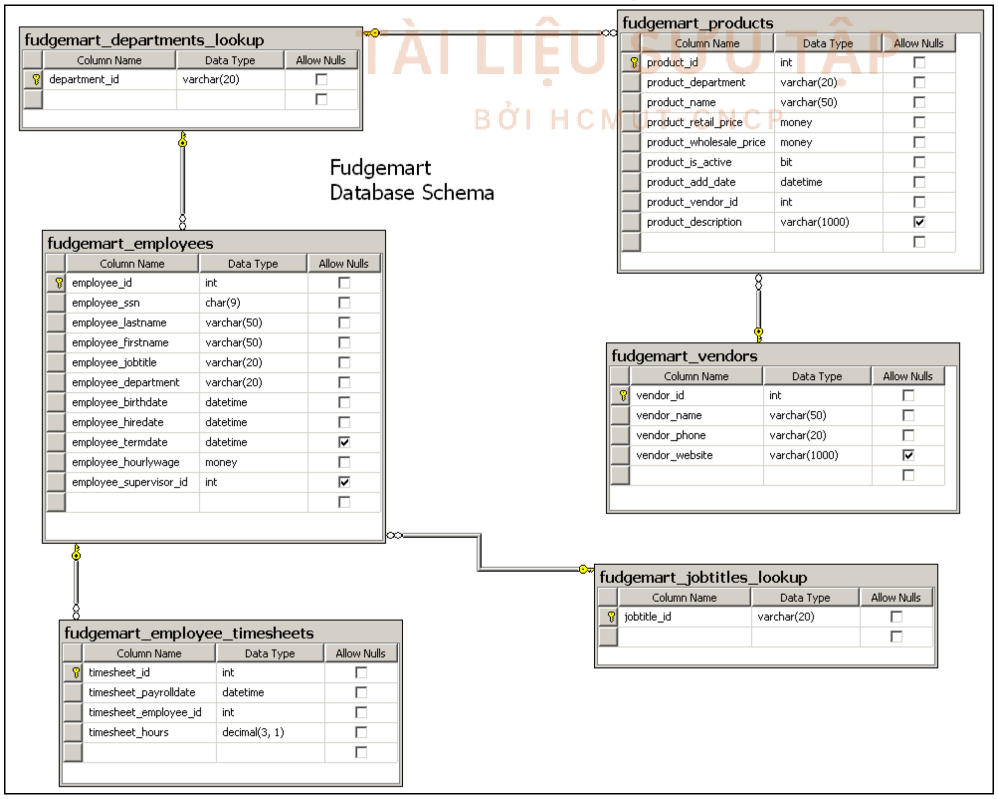
Part 1: The Fudgemart Database Schema

This database supports the business operations of a fictitious mega-store retailer and e-tailer called Fudgemart. The Fudgemart database supports all aspects of the business from human resources, to payroll, to sales transactions, and e-commerce.

1a: The Conceptual Model



1b: Fudgemart Internal Model (Schema)



fudgemart_departments_lookup

Column Name	Data Type	Allow Nulls
department_id	varchar(20)	☐

fudgemart_employees

Column Name	Data Type	Allow Nulls
employee_id	int	☐
employee_ssn	char(9)	☐
employee_lastname	varchar(50)	☐
employee_firstname	varchar(50)	☐
employee_jobtitle	varchar(20)	☐
employee_department	varchar(20)	☐
employee_birthdate	datetime	☐
employee_hiredate	datetime	☐
employee_termdate	datetime	☑
employee_hourlywage	money	☐
employee_supervisor_id	int	☑

fudgemart_employee_timesheets

Column Name	Data Type	Allow Nulls
timesheet_id	int	☐
timesheet_payrolldate	datetime	☐
timesheet_employee_id	int	☐
timesheet_hours	decimal(3, 1)	☐

fudgemart_products

Column Name	Data Type	Allow Nulls
product_id	int	☐
product_department	varchar(20)	☐
product_name	varchar(50)	☐
product_retail_price	money	☐
product_wholesale_price	money	☐
product_is_active	bit	☐
product_add_date	datetime	☐
product_vendor_id	int	☐
product_description	varchar(1000)	☑

fudgemart_vendors

Column Name	Data Type	Allow Nulls
vendor_id	int	☐
vendor_name	varchar(50)	☐
vendor_phone	varchar(20)	☐
vendor_website	varchar(1000)	☑

fudgemart_jobtitles_lookup

Column Name	Data Type	Allow Nulls
jobtitle_id	varchar(20)	☐

1c: Fudgemart External Model

The external data model represents the interface programmers and end-users use to access the data and perform CRUD operations on your database. The following table outlines a subset of the possible external model for the Fudgemart database. Notice how the external model looks more like business processes than anything else. Prefix p_ means Procedure, f_ means function, and v_ means views.

Scope	Task	SQL Object Name
Fudgemart Employees	Add new employee	p_fudgemart_add_new_employee
	Update employee	p_ fudgemart_update_employee
	Alter payrate	p_fudgemart_alter_payrate
	Terminate employee	p_fudgemart_terminate_employee
	Total Employee Hours Worked	f_fudgemart_total_hours_worked
	Display Active Managers	v_ fudgemart_active_managers
	Display Manager's Direct reports	p_fudgemart_get_managers_direct_reports
Fudgemart Timesheets	Add Weekly Timesheet	p_fudgemart_add_timesheet
	Remove Weekly Timesheet	p_fudgemart_remove_timesheet
	Display weekly Timesheet	p_fudgemart_display_weekly_timesheet
	Display annual timesheets (for an employee)	p_fudgemart_display_annual_timesheets
Fudgemart Products	Add new Product	p_fudgemart_add_new_product
	Add new Product and Vendor	p_fudgemart_add_new_product_vendor
	Update Product	p_fudgemart_update_product
	Change Retail Price	p_fudgemart_change_retail_price
	Delete product	p_fudgemart_delete_product
	Deactiveate product	p_fudgemart_deactivate_product
	Display active products	v_fudgemart_display_active_products
	Display vendor products	v_fudgemart_display_vendor_products
Fudgemart Vendors	Vendor Product Count	f_fudgemart_vendor_product_count

NOTE: DO **NOT** ATTEMPT TO CREATE THE EXTERNAL MODEL AT THIS TIME. THIS WILL BE DONE IN LATER PORTIONS OF THE LAB.

Part 2: Create the object, and then use it.

In this part you will first create the SQL object specified, and then write SQL which demonstrates use of the object.

2.a) Execute this code to create the procedure

```
CREATE PROCEDURE p_fudgemart_add_new_employee
        @id int,
        @ssn char(9),
        @lastname varchar(50),
        @firstname varchar(50),
        @jobtitle varchar(20),
        @department varchar(20),
        @birthdate datetime,
        @hiredate datetime,
        @hourlywage money,
        @supervisor_id int
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    IF EXISTS(SELECT * FROM fudgemart_employees WHERE employee_id=@id) RETURN 0
    IF EXISTS(SELECT * FROM fudgemart_employees WHERE employee_ssn=@ssn) RETURN 0
    INSERT INTO fudgemart_employees(
        employee_id, employee_ssn, employee_lastname, employee_firstname,
        employee_jobtitle, employee_department, employee_birthdate,
        employee_hiredate, employee_hourlywage,
        employee_supervisor_id, employee_termdate
    ) VALUES (
        @id, @ssn, @lastname, @firstname,
        @jobtitle, @department, @birthdate,
        @hiredate, @hourlywage, @supervisor_id,
        NULL
    )
    RETURN @@ROWCOUNT
END
```

// @@ROWCOUNT will return the # of rows affected by the last SQL statement to execute

Demonstrate use of this procedure by calling the execute statement.

```
exec p_fudgemart_add_new_employee 40, 189563269, 'Bunn', 'Thomas', 'Department
Manager', 'Electronics', '06/16/1982', '12/01/2008', 20.00, 32
```

2.b) Execute this code to create the procedure

```
CREATE PROCEDURE p_fudgemart_alter_payrate
        @amount decimal(5,2),
        @ispercentage bit
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    IF @ispercentage = 1
    BEGIN
        UPDATE fudgemart_employees
            SET employee_hourlywage = (1 + @amount) * employee_hourlywage
    END
    ELSE
    BEGIN
        UPDATE fudgemart_employees
            SET employee_hourlywage = @amount  +  employee_hourlywage
    END
    RETURN @@ROWCOUNT
END
```

Then write SQL that uses the stored procedure to give everyone a $0.75 raise.

```
exec p_fudgemart_alter_payrate .75, 0
```

2.c) Use the stored procedure you created in 2.b to drop everyone's pay by 5%. Remember 5% = 0.05

```
exec p_fudgemart_alter_payrate -.05, 1
```

2.d) Execute this code to create the following function:

```
CREATE FUNCTION f_fudgemart_total_hours_worked
(
    @id int
)
RETURNS decimal(18,4)
AS
BEGIN
    DECLARE @Result as decimal(18,4)

    SET @Result =  (SELECT SUM(timesheet_hours)
        FROM fudgemart_employee_timesheets
        WHERE timesheet_employee_id=@id)

    -- Return the result of the function
    RETURN @Result

END
```

Write an SQL SELECT statement which displays each employee's name, payrate, and total hours worked using this function.

```
SELECT employee_firstname + ' ' + employee_lastname as 'employee name',
employee_hourlywage, timesheet_hours,
dbo.f_fudgemart_total_hours_worked(employee_id)
FROM fudgemart_employees join fudgemart_employee_timesheets on
employee_id=timesheet_employee_id
```

2.e) Using the function you created in 2.d, write an ALTER TABLE statement to add a column called

employee_total_hours which is a calculated column. Note: to create a calculated column in a table definition, use the syntax column_name AS expression

```
alter table dbo.fudgemart_employees
      add employee_total_hours as
dbo.f_fudgemart_total_hours_worked(employee_id)
```

2.f) Execute this SQL code to create a view:

```
 CREATE VIEW v_fudgemart_active_managers
 AS
     SELECT * FROM fudgemart_employees
         WHERE   employee_termdate is null AND
                 employee_jobtitle <> 'Sales Associate'
```

Using this view, write an SQL select statement to display employee names, and hourly wages of only those managers in the Customer Service department, sorted by hourly wage in ascending order.

```
SELECT employee_firstname + ' ' + employee_lastname as 'employee name',
employee_hourlywage
FROM v_fudgemart_active_managers
WHERE employee_jobtitle <> 'Sales Associate' AND employee_department=
'Customer Service'
ORDER BY employee_hourlywage
```

2.g) Execute the following SQL code to create this stored procedure:

```
 CREATE PROCEDURE p_fudgemart_display_weekly_timesheet
     @week datetime
 AS
 BEGIN
     SELECT  employee_snn, employee_lastname, employee_firstname, employee_department,
             employee_hourlywage, timesheet_hours, timesheet_payrolldate
         FROM fudgemart_employee JOIN fudgemart_employee_timesheets
             ON employee_id = timesheet_employee_id
         WHERE timesheet_payrolldate=@week
 END
```

Then write an SQL statement that uses the procedure to display the timesheet for the week of 1/6/2006

```
exec p_fudgemart_display_weekly_timesheet '1/06/2006'
```

2.h) Execute the following SQL code to create this stored procedure:

```
 CREATE PROCEDURE p_fudgemart_delete_vendor
     @id int
 AS
 BEGIN

     -- need to do this to satisfy referential integrity
     IF EXISTS(SELECT * FROM fudgemart_products WHERE product_vendor_id=@id)
     BEGIN
         DELETE FROM fudgemart_products WHERE product_vendor_id=@id
     END

     DELETE FROM fudgemart_vendors WHERE vendor_id=@id
 END
```

Then write an SQL statement that uses the procedure to delete the vendor ‘Fudgeman’

```
declare @id int;
set @id = (select vendor_id
               from fudgemart_vendors
               where vendor_name = 'Fudgeman')
exec p_fudgemart_delete_vendor @id
```

Part 3: Write the SQL statement which best corresponds to the provided text description

3.a) Write an SQL view called **v_fudgemart_display_active_products** which displays all columns from fudgemart_products where the product is active. It should display the vendor name and phone number for each product as well. Be sure to run a sample SELECT statement demonstrating use of the view.

```
CREATE VIEW v_fudgemart_display_active_products
AS
      SELECT product_id, product_department, product_name,
product_retail_price, product_wholesale_price, product_is_active,
product_add_date, product_vendor_id, product_description, vendor_name,
vendor_phone
      FROM fudgemart_vendors join fudgemart_products on
      vendor_id = product_vendor_id
      WHERE product_is_active = 'True'


Sample select statement

SELECT product_name, product_wholesale_price
FROM v_fudgemart_display_active_products
WHERE product_is_active = 'True' AND vendor_name = 'Leaveeyes'
```

3.b) Write an SQL stored procedure called **p_fudgemart_get_managers_direct_reports** which takes an input an employee ID, and returns the list of names, ssn, and jobtitles of those employees who directly report to that employee ID. (That is the employees where the input parameter is the manager's id.) Be sure to include an exec statement demonstrating use of the procedure.

```
CREATE PROCEDURE p_fudgemart_get_managers_direct_reports
            @empid int

AS
BEGIN
            (SELECT employee_firstname + ' ' + employee_lastname as 'employee
name', employee_ssn, employee_jobtitle
            FROM fudgemart_employees
            WHERE employee_supervisor_id=@empid)
END


exec p_fudgemart_get_managers_direct_reports 32
```

3.c) Write an SQL stored procedure called **p_fudgemart_update_employee** which takes all columns from the fudgemart_employees table (except employee_termdate) as input, and then updates the row with the input parameters for that employee_id. The procedure should return 0 if the employee does not exist.

```
ALTER PROCEDURE p_fudgemart_update_employee
            @empid int,
            @lastname varchar(50),
            @firstname varchar(50),
            @jobtitle varchar(20),
            @department varchar(20),
            @birthdate datetime,
            @hiredate datetime,
            @hourlywage money,
            @supervisor_id int

AS
BEGIN
            SET NOCOUNT ON;
            IF EXISTS(SELECT * FROM fudgemart_employees WHERE
employee_id=@empid)
            UPDATE fudgemart_employees
                  SET employee_lastname = @lastname,
                        employee_firstname = @firstname,
                        employee_jobtitle = @jobtitle,
                        employee_department = @department,
                      employee_birthdate = @birthdate,
                        employee_hiredate = @hiredate,
                        employee_hourlywage = @hourlywage,
                        employee_supervisor_id = @supervisor_id
            RETURN @@ROWCOUNT
END


EXEC p_fudgemart_update_employee 40, 'Bunn', 'Thomas', 'Department Manager',
'Clothing', '06/16/1982', '12/01/2000', 26.75, 32
```

3.d) Write an SQL stored procedure called **p_fudgemart_add_new_product** which inserts a new product into the fudgemart_products table. This procedure should take parameters as input for the data to be inserted.

```
//DIY
```

3.e) Write an SQL stored procedure called **p_fudgemart_deactivate_product** which given a product Id that is currently active, will deactivate that product.

```
//DIY
```

3.f) Write an SQL stored procedure called **p_fudgemart_terminate_employee** which takes an employee id as input and terminates that employee using the current date as the termination date.

```
//DIY
```

3.g) Write an SQL function called **f_fudgemart_vendor_product_count** which given a vendor id returns the number of products that vendor supplies to Fudgemart.

```
//DIY
```

3.h) Write an SQL stored procedure called **p_fudgemart_delete_product** which given a product id will delete that product from the fudgemart_products table.

```
//DIY
```